DEPARTMENT OF INFORMATICS

TECHNISCHE UNIVERSITÄT MÜNCHEN

Bachelor's Thesis in Informatics

Formalisation of Selected Results from Group Theory

Joseph Thommes

DEPARTMENT OF INFORMATICS

TECHNISCHE UNIVERSITÄT MÜNCHEN

Bachelor's Thesis in Informatics

Formalisation of Selected Results from Group Theory

Formalisierung ausgewählter Resultate der Gruppentheorie

Author:	Joseph Thommes
Supervisor:	Prof. Dr. Tobias Nipkow
Advisor:	Dr. rer. nat. Manuel Eberl
Submission Date:	May 14, 2021

I confirm that this bachelor's thesis in informatics is my own work and I have documented all sources and material used.

Munich, May 14, 2021 Joseph Thommes

Acknowledgments

I want to thank my advisor Manuel Eberl for several reasons, the first one being that he accepted my request for developing this thesis in the first place – without me having any prior knowledge of Isabelle or theorem proving. But also throughout the period of working on this project, I found in him an advisor who supplies very useful advice at almost any time of day and night with a responsiveness that is second to none. With him, I had great conversations, not only about mathematics, Isabelle and computer science, and I wish him the best for what may come in his life.

Apart from him, I also am really thankful for the effort that my friends, Jan, Niels and Oliver, put into spotting typos and malformed sentences in this thesis and for their great suggestions for improvement. Together with the rest of my best friends and family, I want to thank them all for their constant support, but also for all the fun we had and will have together.

Abstract

This thesis deals with the formalisation of some group-theoretic results in Isabelle/HOL – an interactive theorem prover that machine-checks every proof step.

The results include the long-known and well understood *fundamental theorem of finitely generated abelian groups* characterising the structure of finitely generated abelian groups as a uniquely determined product of cyclic groups. Both the *invariant factor decomposition* and the *primary decomposition* are covered. Additionally, some results from the field of *character groups* are included, as well as work on formalising the *direct product*, the *internal direct product* and more group-theoretic lemmas – on both the newly introduced definitions and on already existing definitions from *HOL-Algebra* – the canonical Isabelle/HOL library on abstract algebra.

Kurzfassung

Diese Arbeit befasst sich mit der Formalisierung einiger gruppentheoretischer Ergebnisse in Isabelle/HOL – einem interaktiven Theorembeweiser, der jeden Beweisschritt maschinell prüft.

Die Ergebnisse umfassen den seit langem bekannten und gut verstandenen *Fundamentalsatz der endlich erzeugten abelschen Gruppen*, der die Struktur endlich erzeugter abelscher Gruppen als eindeutig bestimmtes Produkt zyklischer Gruppen charakterisiert. Sowohl die *Zerlegung in invariante Faktoren* als auch die *Zerlegung in zyklische Gruppen von Primpotenzordung* werden behandelt. Weiterhin sind einige Ergebnisse aus dem Bereich der *Charaktergruppen* enthalten, sowie Arbeiten zur Formalisierung des *direkten Produkts*, des *inneren direkten Produkts* und weiterer gruppentheoretischer Lemmata – sowohl zu den neu eingeführten Definitionen als auch zu bereits existierenden Definitionen aus *HOL-Algebra* – der kanonischen Isabelle/HOL-Bibliothek zu abstrakter Algebra.

Contents

1 Introduction

While reaching back to the very early 19th century and great names such as Évariste Galois, Carl Friedrich Gauß and Felix Klein, the theory of groups forms a part of mathematics that is still a highly researched topic today, with a broad field of applications in many subjects (e.g. in cryptography).

One of the more recent discoveries is the *classification of finite simple groups* in 1981 – a huge combined effort of many mathematicians spanning thousands of pages [1]. Checking such a long and complex proof is at the same time intellectually demanding, time-consuming and prone to mistakes.

A solution to this problem are machine-checked proofs, involving a computer checking all proof steps and logical inferences for correctness. This way, when trusting the program verifying the proof and checking both the stated theorem and the included definitions, the correctness of a proof accepted by the program follows directly.

For a computer to reason about mathematics, at first it must be translated into constructs suitable for computers – mathematics has to be formalised. This formalisation and its feasibility have been of great interest since at least the beginning of the 20th century and *Hilbert's Program* aiming to formalise all of mathematics. However, with Göbel publishing his *incompleteness theorems* [2], stating that any sufficiently strong consistent logical system possesses undecidable theorems that can neither be proved nor disproved, this program turned out to be unattainable.

Nevertheless, as experience has shown, a huge range of mathematics can – under the assumption of certain axioms – be formalised without further problems, so that today there exist several programs, called theorem provers or proof assistants, suitable for formalising mathematics. One of them is *Isabelle*, a proof assistant with strong automation and the support for several logics for reasoning within it, the most widespread choice being Higher Order Logic (HOL).

In this thesis, I used Isabelle/HOL to formalise several results of group theory, extending the spectrum of formalised theorems by the *fundamental theorem of finitely generated abelian groups* and small parts of the theory of *character groups*. The code of this thesis is available on both zenodo[1] and github[2] and will later be submitted to the *Archive of Formal Proofs*.

The *next chapter* gives a short introduction to Isabelle and HOL-Algebra and introduces some important definitions and notations used throughout this thesis. Furthermore, a short overview of group theory in Isabelle/HOL, Lean, Coq and Mizar is given.

[1] `https://doi.org/10.5281/zenodo.4744644`
[2] `https://github.com/jthomme1/group-theory-isabelle`

Chapter 3 provides an overview of some group-theoretic notions and their formalised equivalent.

In *chapter 4*, I include the formalised proof for the *fundamental theorem of finitely generated abelian groups* in both its forms, as well as a section about the difficulties I encountered during the formalisation of this theorem.

Chapter 5 deals with the application of the *fundamental theorem of finitely generated abelian groups* in the context of *character groups* to prove, among other things, the isomorphism of a group and its character group.

The *last chapter* summarises the results of this thesis and gives an outlook on future work.

2 Preliminaries

All of this work has been formalised in Isabelle/HOL 2021 [3].

2.1 Notation

- Throughout this thesis, I always write groups in a multiplicative way, including abelian groups, using $\otimes$ inside of Isabelle listings and $\cdot$ outside.
- I refer to the *order* of a group G as $|G|$. If it is infinite, the order is by convention 0.
- In an Isabelle listing, whenever the underscore _ is used as a variable name, it indicates an anonymous variable with no name, similar to many programming languages.
- In order to denote the corresponding group in an ambiguous context, the group is put in a subscript, e.g. $\mathbf{1}_G$ for the neutral element of group G.

2.2 The Isabelle proof assistant

> "Isabelle is a generic proof assistant. It allows mathematical formulas to be expressed in a formal language and provides tools for proving those formulas in a logical calculus." ([4])

This quote states that it is possible to prove or disprove the correctness of a formula expressed in a formal language understood by Isabelle. The most common choice is HOL (Higher Order Logic) and this standard combination is then referred to as Isabelle/HOL[1]

This capability of Isabelle/HOL can then be used to formally prove the correctness of, for example, programs or – as in my case – mathematics. However, Isabelle/HOL is not the only proof assistant offering this functionality: Coq[2], Lean[3] and Mizar[4] have similar capabilities, posing the question about the advantages and disadvantages between these different systems and what sets Isabelle/HOL apart.

One important aspect of the answer to this is the *Isar*-language [5] that – opposed to the script-like proving in many other systems – allows the user to formulate their proof

[1]For reasons of convenience, I may refer to *Isabelle/HOL* as *Isabelle* in this work.
[2]`https://coq.inria.fr/`
[3]`https://leanprover.github.io/`
[4]`http://mizar.org/`

in a manner that makes it easier to maintain, more clearly structured and thus easier to understand for the human reader – as it resembles a bit more the classical proof on paper:

```
lemma (in group) pow_int_mod_ord:
  assumes [simp]:"a ∈ carrier G" "ord a ≠ 0"
  shows "a [^] (n::int) = a [^] (n mod ord a)"
proof -
  obtain q r where d: "q = n div ord a" "r = n mod ord a" "n = q ∗ ord a + r"
    using mod_div_decomp by blast
  hence "a [^] n = (a [^] int (ord a)) [^] q ⊗ a [^] r"
    using assms(1) int_pow_mult int_pow_pow
    by (metis mult_of_nat_commute)
  also have "... = 1 [^] q ⊗ a [^] r"
    by (simp add: int_pow_int)
  also have "... = a [^] r" by simp
  finally show ?thesis using d(2) by blast
qed
```

Listing 2.1: Example of a proof in the *Isar* style. Details in [3]

Another argument in favour of Isabelle is its strong proof automation, indispensably assisting the user in his formalisation. Together with a coupling to a wide range of solvers, bundled by the tool sledgehammer [6], it notably shapes the formalising experience towards more productivity.

2.2.1 Archive of Formal Proofs

The *Archive of Formal Proofs (AFP)* "is a collection of proof libraries, examples, and larger scientific developments, mechanically checked in the theorem prover Isabelle" [7]. Although it is already of respectable size with 597 entries and almost 3,000,000 lines of code [8], it is still very incomplete in comparison to all known mathematics.

In the domain of groups, even though there is some central work available such as Zassenhaus's theorem and the Jordan–Hölder theorem [9], the archive still lacks some essential results like the *fundamental theorem of finitely generated abelian groups* – an important theorem describing the structure of a certain type of groups with applications in, for example *character theory*.

With this work I intend to fill this gap.

2.2.2 Miscellaneous definitions and notation

Brackets around assumptions

One syntactical characteristic of Isabelle is the way to write chained implications:

$$\texttt{A} \Longrightarrow \texttt{B} \Longrightarrow \texttt{C} \Longrightarrow \texttt{D} \quad \text{is rewritten as} \quad [\![\texttt{A; B; C}]\!] \Longrightarrow \texttt{D}$$

This is done to highlight the fact that all `A`, `B` and `C` are preconditions for `D` to be true. However, this is just additional syntax and does not change the meaning of the chained implication in any way. The traditional syntax is supported equally.

The `locale` keyword

```
locale monoid =
  fixes G (structure)
  assumes m_closed:
        "⟦x ∈ carrier G; y ∈ carrier G⟧ ⟹ x ⊗ y ∈ carrier G"
  ...

locale group = monoid +
  assumes Units: "carrier G ⊆ Units G"

lemma (in monoid) inv_one: "inv 1 = 1"
```

The `locale` keyword [10] allows to fix the subsequent assumptions and variable names and to bind this context to the name of the locale [11].

So, for example, every `lemma` in the context of the locale `monoid` (denoted by `(in monoid)` after the `lemma` keyword) has the variable `G` available in its scope and can reason about it as if it has been fixed locally. Furthermore, all lemmas from `monoid` are also available in the locale `group`, as it is a `monoid` with the additional assumption that all elements are invertible. This kind of abstraction allows to work in Isabelle efficiently with constructs such as algebraic structures: as just seen, a group is just a monoid where all elements are invertible. As a consequence, all lemmas and theorems about monoids are also true for groups – and when using locales, Isabelle automatically makes these facts easily accessible in the current working context.

`Pi`, `extensional`, `Pi`$_E$ and `restrict`

These four definitions are put into a single section as they are closely related to each other:

```
definition Pi :: "'a set ⇒ ('a ⇒ 'b set) ⇒ ('a ⇒ 'b) set"
  where "Pi A B = {f. ∀x. x ∈ A ⟶ f x ∈ B x}"

definition extensional :: "'a set ⇒ ('a ⇒ 'b) set"
  where "extensional A = {f. ∀x. x ∉ A ⟶ f x = undefined}"

definition "restrict" :: "('a ⇒ 'b) ⇒ 'a set ⇒ 'a ⇒ 'b"
  where "restrict f A = (λx. if x ∈ A then f x else undefined)"

definition PiE :: "'a set ⇒ ('a ⇒ 'b set) ⇒ ('a ⇒ 'b) set"
```

```
  where "PiE A B = Pi A B ∩ extensional A"
```

Listing 2.2: Definitions as in `src/HOL/Library/FuncSet.thy`

`Pi A B` simply describes the set or space of all functions that map all elements of `A` onto elements of `B`, while `extensional A` is used to gather all functions that are undefined[5] outside of the domain `A`. As the name implies, this is important to allow for the principle of extensionality: two elements (functions) are the same, if and only if they agree on the function values of all elements. With a domain of interest `A`, the `extensional` operator allows for the following conclusion:

`⟦f ∈ extensional A; g ∈ extensional A; ∀x∈A. f x = g x⟧ ⟹ f = g`

`Pi`$_E$ `A B` is the intersection between `Pi A B` and `extensional A` – so the set of all functions that are only defined on `A` and from there map every element to an element of `B`.

The `restrict` function reduces the domain of a given function `f` to no larger than `A`, so: `restrict f A ∈ extensional A`. It also supports an alternative, shorter syntax: `restrict f A ≡ (λa∈A. f a)`.

The `image`-operator

The image of a set `A` under application of a function `f` is the union of all function values `f x` of elements `x` in `A`. Although this is a well known, easy to understand and by no means unconventional definition, I include it in this list because of the special syntax it received in Isabelle: ``f ` A`` is the image of the set `A` under `f`.

```
definition image :: "('a ⇒ 'b) ⇒ 'a set ⇒ 'b set" (infixr "`" 90)
  where "f ` A = {y. ∃x∈A. y = f x}"
```

Listing 2.3: Definition of the `image`-operator from `src/HOL/Set.thy`

`SOME` - Hilbert's choice operator

HOL uses the axiom of choice and introduces Hilbert's ϵ-operator:

```
axiomatization Eps :: "('a ⇒ bool) ⇒ 'a"
  where someI: "P x ⟹ P (Eps P)"
```

Listing 2.4: Definition from `src/HOL/Hilbert_Choice.thy`

For a syntactically more intuitive use, the `SOME` operator has been introduced as an alternative syntax to the `Eps` notation: `SOME x. P ≡ Eps (λx. P)`

Set difference

The difference $A \setminus B$ between two sets A and B is written as `A - B` in Isabelle/HOL.

[5] Functions in HOL have to be total. As a consequence, strictly speaking, the notion of a function value being undefined is not supported. However, for every type, there exists a dummy object named *undefined* that is used to mimic the undefinedness.

List access

In Isabelle, the single elements of a list, can be accessed by the `!` operator. The first element of a list `l` can be written as follows: `l!0`. This operation is only well-defined if the length of the list is greater than or equal to the index that is being accessed.

2.2.3 The HOL-Algebra library

HOL-Algebra is the canonical library for abstract algebra in Isabelle/HOL and as such is shipped with the standard installation of Isabelle. In it, one can find the definitions of all classical algebraic structures, such as groups [12].

2.2.4 Groups in HOL-Algebra

Definition of a group

As this thesis revolves around groups in HOL, it is critical to know how they are defined there. The definition can be found in the source files of HOL-Algebra:

```
locale monoid =
 fixes G (structure)
 assumes m_closed:
      "⟦x ∈ carrier G; y ∈ carrier G⟧ ⟹ x ⊗ y ∈ carrier G"
    and m_assoc:
      "⟦x ∈ carrier G; y ∈ carrier G; z ∈ carrier G⟧
       ⟹ (x ⊗ y) ⊗ z = x ⊗ (y ⊗ z)"
    and one_closed: "𝟭 ∈ carrier G"
    and l_one: "x ∈ carrier G ⟹ 𝟭 ⊗ x = x"
    and r_one: "x ∈ carrier G ⟹ x ⊗ 𝟭 = x"

locale group = monoid +
 assumes Units: "carrier G ⊆ Units G"

locale comm_monoid = monoid +
 assumes m_comm: "⟦x ∈ carrier G; y ∈ carrier G⟧ ⟹ x ⊗ y = y ⊗ x"

locale comm_group = comm_monoid + group
```

Listing 2.5: Groups as defined in `src/HOL/Algebra/Group.thy`

As already explained in *subsubsection 2.2.2*, the `locale` keyword allows to fix the subsequent assumptions and variable names and to bind this context to the name of the locale.

Therefore, every `lemma` in the context of the locale `monoid` has the structure `G` available in its scope. This is also true for the group, as it is defined as being a monoid where all elements are invertible (the definition of `Units` was omitted in this excerpt).

It is also worth mentioning that, although mostly clear from the context, it is not possible to identify a group just with its carrier set in a strictly formal environment. Consequently, in HOL-Algebra, a group is a structure with three attributes:

```
record 'a partial_object =
  carrier :: "'a set"

record 'a monoid = "'a partial_object" +
  mult :: "['a, 'a] ⇒ 'a" (infixl "⊗ı" 70)
  one :: 'a ("𝟭ı")
```

Listing 2.6: The structure of a `monoid` as defined in `src/HOL/Algebra/Group.thy` and `src/HOL/Algebra/Congruence.thy`

A set, where the elements of the groups are from: the `carrier`; a binary operation, that combines two elements to a third one, called `mult` (written as ⊗); and the neutral element of this operation, the **1**.

`subgroup` and `generate`

While `subgroup H G` states the fact that `H` is a subgroup of group `G`, `generate G A` describes the subgroup generated by the set `A` in the group `G` and the statement `A ⊆ carrier G ⟹ subgroup (generate G A) G` holds. `generate` is used in several definitions that play a rather central role in this thesis:

```
inductive_set generate :: "('a, 'b) monoid_scheme ⇒ 'a set ⇒ 'a set"
  for G and H where
    one: "𝟭G ∈ generate G H"
  | incl: "h ∈ H ⟹ h ∈ generate G H"
  | inv: "h ∈ H ⟹ invG h ∈ generate G H"
  | eng: "h1 ∈ generate G H ⟹ h2 ∈ generate G H ⟹ h1 ⊗G h2 ∈ generate G H"
```

Listing 2.7: Definition of `generate` from `/src/HOL/Algebra/Generated_Groups.thy`

A `subgroup` in HOL-Algebra – also the one generated by the `generate` function – is not a fully fledged group, but rather just a set with some special properties in the group enclosing it. To turn this set into a fully fledged group, one could overwrite the carrier of the whole group, as the multiplication and neutral element stay the same:

```
G⦇carrier := generate G A⦈
```

Notably, the notion of a subgroup – a group within another group – also proves useful when formalising facts about groups. As stated by Gonthier et al. [13], this is especially beneficial in the case of reasoning about two different groups: it is always possible to view both these groups as subgroups of a bigger group[6] – thus having the same group operation and saving a syntactical and semantical differentiation between the two group

[6] This can be the direct product of the two groups.

operations. Even in the case of a single group, this trick can be of use, as I noticed in the proof of the *fundamental theorem of finitely generated abelian groups*: both the group appearing in the induction hypothesis and the group of interest are subgroups of the same group, removing the need to "translate" between them.

`finprod`

The finite product operator is used to form the product over a (finite) family of elements in a group and can be denoted using the $\bigotimes$-notation. For clarification, let $A = \{a, b, \ldots, z\}$ the index set, G a group and f a function $f : A \to G$. Then:

```
finprod G f A = ⨂x∈A. f x = f a ⊗ f b ⊗ … ⊗ f z
```

As a set has no inherit order to it, the above line could also read `f b ⊗ f a` instead of `f a ⊗ f b`. This implies that the definition of `finprod` only makes sense in the context of a commutative monoid.

I will not go further into details of the definition of `finprod` as it contains some necessary, but for my purposes not important, technicalities[7].

2.3 Related work

2.3.1 Group Theory in Isabelle/HOL

The maintainers of Isabelle have introduced the *Archive of Formal Proofs* and together with the *HOL-Algebra library* in the Isabelle distribution much of the work formalised in Isabelle/HOL is bundled in two central places, including group theory. The work on group theory in HOL-Algebra consists of quotient groups, a binary direct group product, Sylow's Theorem, group actions, the Zassenhaus lemma as well as some work on solvable and symmetric groups [12].

Concerning the *Archive of Formal Proofs*, it includes, inter alia, work on the representation of finite groups [14] and on character groups [15].

The *fundamental theorem of finitely generated abelian groups* however, has, to my knowledge, not yet been formalised – neither in Isabelle/HOL nor in any other proof assistant/theorem prover.

2.3.2 Group Theory in other proof assistants

Of course, the subject of group theory has also been covered in other theorem provers, as it plays a fundamental role in many fields of mathematics. I present here a small overview of the status of group theory in the proof assistants Lean, Coq and Mizar.

[7]These arise from the fact that a set is unordered and one has to somehow "iterate" over it to form the product of all elements.

Lean

Lean has a central library, where it bundles formalisation projects, the *Lean Mathematical Library* [16].

However, it appears to only have rudimentary content on group theory with 4,191 lines of code covering results such as (among others) group actions, quotient groups, free groups, symmetric groups and Galois theory. Results include Sylow's first theorem as well as the Abel–Ruffini theorem.

Coq

The Coq proof assistant caused a stir when in 2013, after a collaborative effort of six years, the Feit–Thompson theorem[8], that played a central role in the classification of finite simple groups, had been formalised using Coq[13]. During this effort, many results on both finite group theory and *character groups* have been formalised.

Like Isabelle and Lean, Coq also has a central place to find and access libraries: the Coq package index. There, the only bigger library for algebra and group theory is *The Mathematical Components repository* with a large section focusing on finite groups and formalising, among others, the Jordan–Hölder theorem [17].

Mizar

Mizar, a theorem prover project initiated in 1973, was among the first to seriously establish the concept of a human-readable but machine-checked proof. This Mizar language inspired, inter alia, the development of the *Isar*-language in Isabelle [5]. Formalisations in Mizar are organised in articles that are bundled in the *Mizar Mathematical Library* and published in the associated journal *Formalized Mathematics* [18]. As of today, the library spans more than 60,000 theorems, in more than 1,300 articles [19]. The group-theoretic work includes results and definitions such as quotient groups, the isomorphism theorems, symmetric groups, the direct product, solvable groups, the Jordan–Hölder theorem, the Sylow theorems and Cayley's theorem. Notably, also the fundamental theorem of finite abelian groups is included, a weaker version of the *fundamental theorem of finitely generated abelian groups* [20].

However, in all mentioned theorem provers and their respective libraries, there is no specific work on finitely generated abelian groups or their decomposition.

[8]also known as the odd order theorem

3 Formalisation Decisions

3.1 Group locales

In the course of this thesis, I had to reason about different types of groups. A situation where the concept of *locales* is useful. The following listing shows the different group locales introduced in my work:

```
locale finite_group = group +
  assumes fin: "finite (carrier G)"

locale cyclic_group = group +
  fixes gen :: "'a"
  assumes gen_closed: "gen ∈ carrier G"
  assumes generator: "carrier G = generate G {gen}"

locale finite_comm_group = finite_group + comm_group

locale finite_cyclic_group = finite_group + cyclic_group

locale fin_gen_comm_group = comm_group +
  fixes gen :: "'a set"
  assumes gens_closed: "gen ⊆ carrier G"
  and fin_gen: "finite gen"
  and generators: "carrier G = generate G gen"
```

Clearly, there are connections between these types of groups that are not directly reflected by inheritance – for example, that all cyclic groups are commutative. These facts have been proven manually:

```
sublocale cyclic_group ⊆ comm_group

sublocale finite_comm_group ⊆ fin_gen_comm_group G "carrier G"
```

3.2 `IDirProds` and `is_idirprod` – internal direct product

An important notion in the context of a decomposition of groups, as done by the *fundamental theorem of finitely generated abelian groups*, is the *internal direct product.*

A group is called the *internal direct product* of a set of subgroups Hs of the group G if the following three conditions are satisfied [21]:

1. Every subgroup $H \in Hs$ is normal ($\forall g \in G.\ g \cdot H \cdot g^{-1} = H$, trivially true for commutative groups).

2. All $H \in Hs$ together generate G.

3. For every subgroup $H \in Hs$ it holds that the intersection of H and the subgroup generated by $Hs \setminus \{H\}$ is trivial – i.e. just the neutral element $\mathbf{1}$.

In Isabelle/HOL, these can be formalised like the following:

1. `∀H ∈ Hs. H ◁ G`

2. `carrier G = generate G (⋃Hs)`

For this, I have introduced the following definition:

```
definition IDirProds :: "('a, 'b) monoid_scheme ⇒ 'a set set ⇒ 'a set"
  where
  "IDirProds G Hs = generate G (⋃Hs)"
```

3. `∀H ∈ Hs. complementary H (IDirProds G (Hs - H))` with `complementary H1 H2 ⟷ H1 ∩ H2 = 1`.

Thus, the subsequent definition:

```
definition (in group) complementary_family :: "'a set set ⇒ bool" where
  "complementary_family Hs
  = (∀H ∈ Hs. complementary H (IDirProds G (Hs - {H})))"
```

These three formalisations are directly reflected in the following definition to characterise an internal direct product[1]:

```
inductive (in group) is_idirprod :: "'a set ⇒ 'a set set ⇒ bool" where
  "(⋀H. H ∈ Hs ⟹ H ◁ G) ⟹
   A = IDirProds G Hs ⟹
   complementary_family Hs ⟹
   is_idirprod A Hs"
```

[1]The only reason that this has been defined using the `inductive` keyword instead of the standard `definition`, is that it introduces a greater amount of predefined simplification rules about the predicate. Semantically, one could also have used `definition`.

3.2.1 compl_gens and is_idirgen

In the course of the proof of the *fundamental theorem of finitely generated abelian groups*, the need arose to reason not only about a set of subgroups, but about a set of elements (generators), each generating a subgroup by itself.

With this slightly different setting, I also needed a different predicate expressing essentially the same as point **3** of the preceding paragraph, but talking about a set of single elements `A`. This is because the naïve approach of just checking for `is_idirprod ((λg. generate G {g}) ` A)` does not capture the intended meaning in its entirety, as it would also allow for several different elements creating the same subgroup to be in `A`.

There are several ways to enforce the intended behaviour: The first one is to just explicitly force the injectivity of `(λg. generate G {g})` on `A` by, for example: `inj_on (generate G) A`.

I, however, chose another approach: I added another predicate that expresses the combined meaning of `complementary_family` and the aforementioned injectivity:

```
definition (in group) compl_gens :: "'a set ⇒ bool" where
  "compl_gens gs
  = (∀g ∈ gs. complementary (generate G {g}) (generate G (gs - {g})))"
```

Here, `generate` is used instead of `IDirProds` as we are acting on a set of elements rather than a set of sets of elements as when dealing with subgroups in `complementary_family`. To justify this definition, I proved that it provided just what I intended:

```
lemma (in group) compl_gens_imp_complementary_family:
  assumes "gs ⊆ carrier G" "compl_gens gs"
  shows "complementary_family ((λg. generate G {g}) ` gs)"

lemma (in group) compl_gens_imp_generate_inj:
  assumes "gs ⊆ carrier G" "compl_gens gs"
  shows "inj_on (λg. generate G {g}) gs"
```

To make working with this new definition a bit more convenient, I also defined `is_idirgen`:

```
inductive (in group) is_idirgen :: "'a set ⇒ 'a set ⇒ bool" where
  "(⋀g. g ∈ gs ⟹ (generate G {g}) ⊲ G) ⟹
   A = generate G gs ⟹
   compl_gens gs ⟹
   is_idirgen A gs"
```

Listing 3.1: Definition of `is_idirgen`

With the lemmas showed before, it was easy to derive the following:

```
lemma (in group) idirgen_imp_idirprod:
  assumes "is_idirgen A gs" "gs ⊆ carrier G"
  shows "is_idirprod A ((λg. generate G {g}) ` gs)"
```

This lemma validates the definition of `is_idirgen`, as it implies `is_idirprod` for the subgroups generated by all the single elements – which is just what will be needed.

3.3 `DirProds` – (external) direct product

The most central notion when reasoning about decomposing groups is the *(external) direct product* of groups.

The direct product of groups G_i with $i \in \mathbb{N}$ is defined [21] as:

$$G_0 \times G_1 \times G_2 \ldots = \{(g_0, g_1, g_2 \ldots) \mid g_i \in G_i\}$$

with the operation between two elements defined component-wise:

$$(g_0, g_1, g_2 \ldots) \cdot (h_0, h_1, h_2 \ldots) = (g_0 \cdot h_0, g_1 \cdot h_1, g_2 \cdot h_2 \ldots)$$

One way to formalise this would be to use list as elements of the direct product group, where the components of the element directly correspond to the single components of the list. This would be fine as long as we just want to represent a finite product.

For an infinite number of factors however, this formalisation choice would not be able to represent the product, as lists are finite in Isabelle. To also allow for infinite products, the elements of the product group were chosen to be functions that for each index in a (possibly) infinite index set return a component of the element. This decision of using index sets instead of lists brings the benefit of not introducing the illusory image of an order in the product and so, things like its ”commutativity” are trivialities, that, in the case of a list, would have to be proven.

So, the definition of the direct product in Isabelle/HOL is the following[2]:

```
definition DirProds
        :: "('a ⇒ ('b, 'c) monoid_scheme) ⇒ 'a set ⇒ ('a ⇒ 'b) monoid"
  where
  "DirProds G I = ⦇ carrier = Pi_E I (carrier ∘ G),
             mult = (λx y. restrict (λi. x i ⊗_G i y i) I),
             one = restrict (λi. 𝟭_G i) I ⦈"
```

`carrier = Pi_E I (carrier ∘ G)` encodes that each element of the direct product should have its components in the `carrier` of the corresponding group for every index in the index set `I` while the element (which is a function) should be `undefined` for all indices that are not in `I` to allow for extensionality, i.e. two elements are equal iff they agree on

[2]Note that the elements of the direct product are not restricted to a finite number of non-trivial components, so that this is indeed the direct product (product in category theory) – not the direct sum (coproduct in category theory). As these two notions coincide for finitely many factors and as terminology is not too consistent with this concept, they are often used interchangeably depending on the context. However, as this thesis mainly focuses on finitely generated groups, where only a finite number of factors occurs in a direct product, this does not matter too much in the context of my work.

all components. This is represented by the predicate `extensional I` which is used in the definition of $\texttt{Pi}_E$[3].

The multiplication is defined component-wise with the additional requirement that the combination of two elements has to be `undefined` outside of the index set `I` because of extensionality, which is encoded by `restrict f I` for a function `f`.

It is quite obvious that the neutral element of the direct product is the function returning the neutral element of each of the groups corresponding to the indices in the index set `I`.

The reason to be interested in the direct product at all is that it itself forms a group, which can easily be shown:

```
lemma DirProds_group_iff: "group (DirProds G I) ⟷ (∀i∈I. group (G i))"


lemma DirProds_comm_group_iff:
  "comm_group (DirProds G I) ⟷ (∀i∈I. comm_group (G i))"
```

As the names *direct product* and *internal direct product* suggest, the two notions are related: if a group is the internal product of a finite set of subgroups, then it is isomorphic to the direct product of these subgroups. In Isabelle:

```
lemma (in comm_group) cong_DirProds_IDirProds:
  assumes "is_idirprod (carrier G) Hs" "finite Hs"
  shows "DirProds (λH. G⦇carrier := H⦈) Hs ≅ G"
```

Listing 3.2: Isomorphism of Direct Product and Internal Direct Product

3.4 Miscellaneous Definitions

3.4.1 `relations` of elements

The proof of the *fundamental theorem of finitely generated abelian groups* requires the notion of a relation between elements. While not particularly difficult, it is worth an explanation and a proper introduction.

Informally, a relation of a set of elements is a way to express the neutral element using powers of these elements.

In my formalisation efforts, I defined the set of all relations for the subgroup generated by `A` as follows[4]:

```
definition (in comm_group) relations :: "'a set ⇒ ('a ⇒ int) set" where
  "relations A = {f. finprod G (λa. a [^] f a) A = 1} ∩ extensional A"
```

`[^]` is the exponentiation operator within the group – i.e. exponentiation using the multiplication function of the group. A relation is then a function that assigns to each element a_i of the set A an integer exponent e_i, so that the term $\prod_{i=1}^{n} a_i^{e_i}$ equals $\mathbf{1}$.

[3]cf. *subsubsection 2.2.2* `Pi`*,* `extensional`*,* $\texttt{Pi}_E$ *and* `restrict`

[4]The reason why this is only defined for commutative groups is explained in *subsubsection 2.2.4* `finprod`.

Note that in the proof of the *fundamental theorem of finitely generated abelian groups*, this notion is also used, but with a subtle difference: Here, the elements are considered a set – and in the just mentioned proof, the elements are taken as a list. This very small difference does matter in the formal proof and will be explained in *section 4.4 Formalisation Decisions and Difficulties.*

3.4.2 `get_exp` – resembling the discrete logarithm

The `get_exp` function is defined as follows: `get_exp g a` returns an exponent e so that $g^e = a$. This is obviously only possible if there exists such an exponent. Even then, the choice is in most cases not unique. As a consequence, the *choice operator* has been used to formalise this function:

```
definition (in group) get_exp where
  "get_exp g = (λa. SOME k::int. a = g [^] k)"
```

This definition, although not mentioned a lot throughout this thesis, has proved useful at several occasions throughout my formalisation work.

4 Fundamental theorem of finitely generated abelian groups

A main focus of this thesis is the *fundamental theorem of finitely generated abelian groups* – a theorem well known in today's form since the beginning of the 20th century and with proofs for weaker versions and special cases going back to the early 19th century [22]. The theorem can be formulated in two equivalent versions.

4.1 Invariant factor decomposition

Any finitely generated abelian group G can be decomposed into a direct product of cyclic groups whose orders divide each other successively:

$$G \cong \mathbb{Z}_{o_1} \times \ldots \times \mathbb{Z}_{o_k} \quad \text{with} \quad \forall i \in \{1, \ldots k-1\}. \quad o_i \mid o_{i+1}$$

Note: The order of $\mathbb{Z}$ is by convention 0, so $\mathbb{Z}_0 = \mathbb{Z}$. Moreover, as every natural number divides 0, but the only number to divide 0 is 0 itself, the $\mathbb{Z}$ components are the last factors of the product. This convention is not arbitrary – it is also used in HOL-Algebra.

This decomposition is uniquely determined by G and the $o_i \neq 0$ are called the *invariant factors* of G, giving this version of the theorem its name. This decomposition is the coarsest one of the group (least number of cyclic factors).

4.2 Primary decomposition

The primary decomposition provides the finest decomposition of a finitely generated abelian group G into cyclic groups. G is decomposed into a product of *primary cyclic groups* – that is, cyclic groups whose order is a prime power or infinite:

$$G \cong \mathbb{Z}_{q_1} \times \ldots \times \mathbb{Z}_{q_k} \quad \text{where the } q_i \text{ are prime powers or 0.}$$

Note: The primes do not need to be distinct.

This decomposition is unique up to permutation of the factors.

4.3 Proof

4.3.1 Invariant factor decomposition

The proof I chose to formalise is the one Kemper provides for the *invariant factor decomposition* in the lecture notes to his undergraduate algebra lecture [23].
What now follows is a slightly modified version of his proof from these notes:

Let G be a finitely generated abelian group, generated by n elements. Then:

$$G \cong \mathbb{Z}_{d_1} \times \ldots \times \mathbb{Z}_{d_k} \quad \text{with} \quad k \leq n \quad \text{and} \quad \forall i \in \{1, \ldots, k-1\}. \quad d_i \mid d_{i+1}$$

Proof: We have $G = \langle \sigma_1, \ldots, \sigma_n \rangle$ and perform a proof by complete induction on n. We introduce the notion of a relation between elements: a relation between n elements σ_i is a product of the following form:

$$\prod_{i=1}^{n} \sigma_i^{e_i} = \mathbf{1},\ e_i \in \mathbb{Z}$$

Note that this is always the case when all $e_i = 0$ (trivial relation).

Firstly, we consider the case that there is no relation between the σ_i besides the trivial one – so that from $\prod_{i=1}^{n} \sigma_i^{e_i} = \mathbf{1},\ e_i \in \mathbb{Z}$ it follows that all $e_i = 0$. In this case, the order of all σ_i is infinite (because if not, $\sigma_i^{ord(\sigma_i)} = \mathbf{1}$ would be a relation) and as there is also no relation "connecting"[1] the generators, we have[2]:

$$G \cong \langle \sigma_1 \rangle \times \ldots \times \langle \sigma_n \rangle \cong \mathbb{Z} \times \ldots \times \mathbb{Z} \quad \text{proving the theorem for this case.}$$

The remaining case is that there are relations between the σ_i, i.e. there exist e_i such that $\prod_{i=1}^{n} \sigma_i^{e_i} = \mathbf{1}$ where not all exponents $e_i = 0$. We choose a relation of n elements α_i that – among all relations of n elements generating the whole group – involves the minimal positive exponent occurring in all of these relations:

$$\prod_{i=1}^{n} \alpha_i^{e_i} = \mathbf{1},\ e_i \in \mathbb{Z}$$

Without loss of generality, let α_1 have this smallest exponent e_1. This minimality is a key fact that will be used in several occasions and I will refer to it as **(M)**.

We can obtain this minimum because there exist relations with positive exponents: any relation with an exponent $e_i < 0$ belonging to the element σ_i can be turned into a relation with a positive exponent by replacing $\sigma_i^{e_i}$ with $(\sigma_i^{-1})^{-e_i}$.

The first step is then to show that all e_i are multiples of e_1: using integer division, we obtain $r, q \in \mathbb{Z},\ 0 \leq r < e_1$, so that $e_i = e_1 \cdot q + r$. With $\tau_1 := \alpha_1 \alpha_i^q$, it holds that:

$$G = \langle \tau_1, \alpha_2, \ldots, \alpha_n \rangle \quad \text{and} \quad \alpha_i^r \cdot \tau_1^{e_1} \cdot \prod_{\substack{1 \leq j \leq n \\ j \neq i}} \alpha_j^{e_j} = \mathbf{1}$$

[1]More details in *section 4.4 Formalisation Decisions and Difficulties*

[2]Here, we use the well-known fact that cyclic groups of the same order are isomorphic.

The first equation holds, since τ_1 is formed using α_1 and α_i by definition and α_1 can be expressed using τ_1 and α_i: $\alpha_1 = \tau_1 \alpha_i^{-q}$.

For the second part, note that:

$$\alpha_i^r \cdot \tau_1^{e_1} \cdot \prod_{\substack{1 \le j \le n \\ j \ne i}} \alpha_j^{e_j} = \alpha_i^r \cdot \alpha_i^{e_1 q} \cdot \alpha_1^{e_1} \cdot \prod_{\substack{1 \le j \le n \\ j \ne i}} \alpha_j^{e_j} = \prod_{j=1}^{n} \alpha_j^{e_j} = \mathbf{1}$$

These elements thus form a relation of n elements, and $r \neq 0$ would be a contradiction to **(M)** (as $r \le e_1$). It follows that $\forall i \in \{1, \dots, n\}.\ e_1 \mid e_i$.

Next, let $\tau := \prod_{i=1}^{n} \alpha_i^{e_i/e_1}$ so that

$$G = \langle \tau, \alpha_2 \dots, \alpha_n \rangle \quad \text{and} \quad \tau^{e_1} = \mathbf{1}$$

From this, it follows that $ord(\tau) \mid e_1$. But, because $\mathbf{1} = \tau^{ord(\tau)} = \tau^{ord(\tau)} \cdot \prod_{i=2}^{n} \alpha_i^0$ is also a relation with n elements, we have, together with **(M)**, that $ord(\tau) = e_1$.

In the case that $e_1 = 1$, it follows that $\tau = \mathbf{1}$ and the theorem follows by the induction hypothesis, as the group is then generated by $a_2, \dots, a_n$.

In the remaining case where $e_1 > 1$, it holds that $\langle \tau \rangle \cap \langle \alpha_2, \dots, \alpha_n \rangle = \{\mathbf{1}\}$ as every element of the intersection can be written both as τ^a with $0 \le a < ord(\tau) = e_1$ and as $\prod_{i=2}^{n} \alpha_i^{a_i}$. This leads to $\tau^a \cdot \prod_{i=2}^{n} \alpha_i^{-a_i} = \mathbf{1}$; a contradiction to **(M)** for $a > 0$. Thus, $a = 0$, and $\mathbf{1}$ is the only element in the intersection, making G by definition the internal direct product of $\langle \tau \rangle$ and $\langle \alpha_2, \dots, \alpha_n \rangle$. And as stated in Listing 3.2, this implies:

$$G \cong \langle \tau \rangle \times \langle \alpha_2, \dots, \alpha_n \rangle \cong \mathbb{Z}_{d_1} \times \langle \alpha_2, \dots, \alpha_n \rangle$$

From the induction hypothesis we get:

$$\langle \alpha_2, \dots, \alpha_n \rangle \cong \mathbb{Z}_{d_2} \times \dots \times \mathbb{Z}_{d_k} \quad \text{with} \quad k \le n \quad \text{and} \quad \forall i \in \{2, \dots, k-1\}.\ \ d_i \mid d_{i+1}$$

It thus remains to show that $d_1 \mid d_2$. In the case that $k < n$, there are less than n generators and we are done by induction. So, we can assume $k = n$. To every $\mathbb{Z}_{d_i}$ there exists a generator τ_i generating it (such that $\tau = \tau_1$). We have the relation $\tau_1^{d_1} \cdot \tau_2^{d_2} \cdot \prod_{i=3}^{n} \tau_i^0 = \mathbf{1}$ of n elements. Using the argument of the integer division like before, we can obtain a contradiction to **(M)** if $d_1 \nmid d_2$, thus showing the theorem. □

The theorem above shows the existence of such a decomposition, but it does not show the uniqueness of it. I let this part open for future work.

4.3.2 Primary decomposition

The *primary decomposition* can directly be obtained from the *invariant factor decomposition* using the fact that for each $n > 0$, it holds that:

$$\mathbb{Z}_n \cong \mathbb{Z}_{p_1^{a_1}} \times \dots \times \mathbb{Z}_{p_m^{a_m}}$$

where the p_i are the prime factors of n and the a_i their respective multiplicities. This is true because the $p_i^{a_i}$ are all coprime to each other (cf. Chinese remainder theorem). □

Here, the uniqueness could be obtained in a similar fashion from the uniqueness of the *invariant factor decomposition*, but as this is missing, the uniqueness of the primary decomposition also remains unproven and open for future work.

4.4 Formalisation Decisions and Difficulties

As the attentive reader may have noticed, the proof of the *invariant factor decomposition* includes a part which is not formulated in a strictly formal way: the fact that there is no "connection" between the generating elements σ_i in case that there is only the trivial relation between them and the subsequent conclusion, that they thus form:

$$G \cong \langle\sigma_1\rangle \times \ldots \times \langle\sigma_n\rangle \cong \mathbb{Z} \times \ldots \times \mathbb{Z}$$

In order to prove this, I first had to show the equivalence of a different way to express the fact that subgroups form an internal direct product:

```
lemma (in comm_group) triv_finprod_iff_comp_fam_PiE:
  assumes "finite Hs" "⋀H. H ∈ Hs ⟹ subgroup H G"
  shows "(∀f ∈ Pi_E Hs id. finprod G f Hs = 𝟭 ⟶ (∀H∈Hs. f H = 𝟭))
        ⟷ complementary_family Hs"
```

This states that a finite complementary family of subgroups is exactly characterised by the fact that, if we pick one element x_i from each subgroup H_i and take the product of all these, this can only be **1** iff all the x_i are **1**.

However, not only the subgroups but also their generators are of interest in the previous proofs, requiring a slightly stronger version of the lemma:

```
lemma (in comm_group) triv_finprod_iff_comp_gens:
  assumes "finite gs" "gs ⊆ carrier G"
  shows
    "(∀f ∈ Pi_E gs (λa. generate G {a}). finprod G f gs = 𝟭 ⟶ (∀a∈gs. f a = 𝟭))
     ⟷ compl_gens gs"
```

With this, it was not difficult to show the following:

```
lemma (in comm_group) comp_fam_iff_relations_triv:
  assumes "finite gs" "gs ⊆ carrier G" "∀g∈gs. ord g = 0"
  shows "relations gs = {(λ_∈gs. 0::int)} ⟷ compl_gens gs"
```

which is just what was proposed in the slightly informal part of the proof.

Apart from that, as I already hinted on in *subsection 3.4.1* `relations` *of elements*, there is a difference between the definition of a relation as used in the previous proof and as formalised in Isabelle. The former considers a list of elements of length n and the latter a set of elements of cardinality n. Thinking of the elements as a set rather than a list, brings a disadvantage: the key fact used was the minimality **(M)** of an exponent of a relation of n elements. Every time that I constructed a different relation to contradict **(M)**, I had to prove that it consists of n elements. In the case of a list, this does not pose a problem as I only ever substituted an element with a different one, thus not changing the length of the list. But in the case of a set, this is not that simple: the new element substituting another could coincide with an element already in the set, thus reducing its cardinality to $n-1$.

A countermeasure to this problem is to introduce another change to the proof: instead of considering relations of *exactly* n elements, I worked with relations of *at most* n elements. However, when proving that there are indeed relations with a positive exponent – by replacing an element with a negative exponent trough its inverse – this change does not prevent having to consider in an extra case that the inverse of the element could already be among the generators.

All of this did not prevent me from proving the theorem with this definition of relations, but in retrospect, I would change the definition of a relation so that it is formulated in terms of a list, to allow for a slightly cleaner proof.

Another technique used in the formalised proof is to not prove the theorem directly for a finitely generated abelian group, but for a finitely generated subgroup of an abelian group. As mentioned in *subsubsection 2.2.4*, this allows for a cleaner induction, as both the group of interest and the group appearing in the induction hypothesis are subgroups of the same group, sharing the group multiplication and neutral element.

4.5 The final formalised theorems

Following the path outlined in *subsection 4.3.1 Invariant factor decomposition*, I ended up with the slightly stronger following result:

```
theorem (in fin_gen_comm_group) invariant_factor_decomposition_idirgen:
  obtains gs where
    "set gs ⊆ carrier G" "distinct gs" "is_idirgen (carrier G) (set gs)"
    "successively (dvd) (map ord gs)" "card (set gs) ≤ card gen" "1 ∉ set gs"
```

Here, it is also stated that the obtained decomposition (encoded in a list of generators `gs`) consists of no more elements than required to generate the group. Moreover, it is excluded that the trivial factor $\mathbb{Z}_1$ occurs in the product[3].

With this decomposition, I was able to formalise the *fundamental theorem of finitely generated abelian groups* in its original way:

```
corollary (in fin_gen_comm_group) invariant_factor_decomposition_Zn:
  obtains ns where
    "DirProds (λn. Z (ns!n)) {..<length ns} ≅ G"
    "successively (dvd) ns" "length ns ≤ card gen"
```

[3]The decomposition of $\mathbb{Z}_1$ itself would be the empty product.

```
corollary (in fin_gen_comm_group) primal_decomposition_Zn:
 obtains ns where
   "DirProds (λn. Z (ns!n)) {..<length ns} ≅ G"
   "∀n∈set ns. n = 0 ∨ (∃p k. prime p ∧ k > 0 ∧ n = p ^ k)"
```

In both cases, `ns` is a list containing the orders of the cyclic integer groups that take part in the product. Note that `Z x` is an abbreviation for `integer_mod_group x`, which in mathematical notation is $\mathbb{Z}_x$.

5 Character groups

The term of a *character group* is a notion that occurs in the contexts of algebra and analytical number theory and has a huge amount of applications, for example when proving *Dirichlet's Theorem on primes in arithmetic progressions* [15]. There are (at least) two different, but related definitions of the term *character*: the *multiplicative character* and the *character of a representation*. In this work however, I exclusively refer to the *multiplicative character*.

I extend and simplify a part of an already existing entry in the *Archive of Formal Proofs* that includes some theory about characters by Eberl [15], by making use of the *fundamental theorem of finitely generated abelian groups* that I have formalised before.

5.1 Character groups

A (multiplicative) character χ on a finite abelian group G is defined as a group homomorphism $\chi : G \to \mathbb{C}^*$ (i.e. $\mathbb{C} \setminus \{0\}$ with the multiplication as a group operation) [24]. This set of homomorphisms forms an abelian group $\widehat{G}$ (the *character group* or G's dual group) with the multiplication of two characters χ_1, χ_2 (extensionally) defined as:

$$(\chi_1 \cdot \chi_2)(a) := \chi_1(a) \cdot \chi_2(a) \quad \text{for} \quad a \in G.$$

Being a group homomorphism, a character must project the neutral element of G onto the neutral element of the multiplication in $\mathbb{C}$: the number 1. With the just mentioned multiplicative property of characters, it follows for an element $a \in G$:

$$1 = \chi(\mathbf{1}) = \chi(a^n) = \chi(a)^n$$

Note: $n := |G|$ here and in the rest of this chapter.

This implies two things:
1. The neutral element – called the *principal character* – of the character group $\widehat{G}$ is just the function that projects every element of the group onto the number 1.
2. Each character is a function that maps the elements of the group G onto one of the n-th roots of unity. The image of a character function thus resides on the unit circle.

5.2 Initial State

The entry in the AFP, that I targeted on restructuring and extending, revolves around *Dirichlet's Theorem on primes in arithmetic progressions* [15] and makes use of the theory of character groups. In the entry, there are several thematically structured theories.

My work focuses on the theory `Multiplicative_Characters`. By restructuring its proofs, it renders superfluous the also included theory `Adjoin_Groups` – a rather ad-hoc appearing theory with the sole purpose of enabling some of the proofs about characters. In this theory file, the author considers a subgroup H of a group G and – without using the standard definition of `generate` – constructs a new subgroup of G by adding (adjoining) another group element to it and closing it under multiplication and the forming of an inverse.

In `Multiplicative_Characters` the notion of characters and character groups for finite abelian groups are introduced. Moreover, it is shown that the character groups themselves form finite abelian groups.

The two orthogonality theorems of characters, the natural isomorphism between a group and its double dual, and some useful facts – such as being able to extend a character on a subgroup to a character on the whole group – also found their way into this file, all making use of the theory `Adjoin_Groups`.

```
locale character = finite_comm_group +
  fixes χ :: "'a ⇒ complex"
  assumes char_one_nz: "χ 1 ≠ 0"
  assumes char_eq_0: "a ∉ carrier G ⟹ χ a = 0"
  assumes char_mult [simp]:
    "⟦a ∈ carrier G; b ∈ carrier G⟧ ⟹ χ (a ⊗ b) = χ a * χ b"

definition principal_char :: "('a, 'b) monoid_scheme ⇒ 'a ⇒ complex" where
  "principal_char G a = (if a ∈ carrier G then 1 else 0)"

definition characters :: "('a, 'b) monoid_scheme ⇒ ('a ⇒ complex) set" where
  "characters G = {χ. character G χ}"

definition Characters :: "('a, 'b) monoid_scheme ⇒ ('a ⇒ complex) monoid"
  where "Characters G = ⦇ carrier = characters G,
                        mult = (λχ1 χ2 k. χ1 k * χ2 k),
                        one = principal_char G ⦈"
```

Listing 5.1: Definitions from the file `Multiplicative_Characters`.

5.3 Contributions

5.3.1 Character group isomorphism

After having done some proofs about abelian and cyclic groups throughout the course of this thesis, I began my work on this section with the special case of cyclic groups following the notes of Evertse and Sofos [24]. This case was a great starting point, since cyclic groups are a very basic structure being completely determined a single generating element. This simplicity and the multiplicative characteristics of a character allow to completely determine a character χ on a group G generated by an element gen by just the value of $\chi(gen)$. This inspired me to define (only locally for the following lemmas) the function `induce_char`[1] that takes a value $\chi(gen)$ and constructs a character on the cyclic group G. I proved that this function actually is a bijection between the nth roots of unity and all the characters on G, implying the results in the following listing:

```
lemma (in finite_cyclic_group)
  defines ic: "induce_char ≡
    (λc::complex. (λa. if a∈carrier G then c powi get_exp gen a else 0))"
  shows order_Characters: "order (Characters G) = order G"
  and gen_fixes_char: "⟦character G a; character G b; a gen = b gen⟧ ⟹ a = b"
  and unity_root_induce_char: "z ^ order G = 1 ⟹ character G (induce_char z)"
```

Listing 5.2: Results on character groups of cyclic groups

It was also possible to show that a single one of these characters generates the whole character group if it maps a generating element of G onto a "true" nth root of unity y (i.e. n is the smallest positive exponent so that $y^n = 1$). This allows to interpret the character group of a finite cyclic group also as a finite cyclic group:

```
lemma (in finite_cyclic_group) finite_cyclic_group_Characters:
  obtains χ where "finite_cyclic_group (Characters G) χ"
```

And as cyclic groups of the same order are isomorphic, it follows directly:

```
lemma (in finite_cyclic_group) Characters_iso:
  "G ≅ Characters G"
```

With this isomorphism of the "building blocks" of finite abelian groups, I had taken the first step in order to prove the isomorphism between a group and its character group. Next, I had to consider characters on a direct product of groups. Here, it was necessary to analyse the connection between a single character on a factor of the group product and the character on the entire product. Evertse and Sofos [24] just define a character χ on a direct product G in terms of the single characters χ_i of the "factor" groups G_i in the product:

$$\chi(g) = \chi(g_1, \dots, g_n) := \prod_{i=1}^{n} \chi_i(g_i)$$

While this definition is correct, one cannot just simply define the character on the direct product in Isabelle/HOL. Instead, it has to be proven that this really is **the** way characters behave on a direct product, i.e. one has to show that every character on a direct product induces character functions on its components and that its value indeed is the product of these component character functions applied to each component. This results in the following rather complex formulations:

```
lemma DirProds_subchar:
  assumes "finite_comm_group (DirProds Gs I)"
```

[1] for the definition of `get_exp` see *subsection 3.4.2* `get_exp` *– resembling the discrete logarithm*

```
  and x: "x ∈ carrier (Characters (DirProds Gs I))"
  and i: "i ∈ I" and I: "finite I"
  defines g: "g ≡ (λc. (λi∈I. (λa. c ((λi∈I. 𝟭Gs i)(i:=a)))))"
  shows "character (Gs i) (g x i)"

lemma Characters_DirProds_single_prod:
  assumes "finite_comm_group (DirProds Gs I)"
  and x: "x ∈ carrier (Characters (DirProds Gs I))"
  and I: "finite I"
  defines g: "g ≡ (λI. (λc. (λi∈I. (λa. c ((λi∈I. 𝟭Gs i)(i:=a))))))"
  shows "(λe. if e∈carrier(DirProds Gs I) then ∏i∈I. (g I x i) (e i) else 0) = x"
```

With these two lemmas I can show the isomorphism of a character group of a direct product to the direct product of character groups:

```
lemma (in finite_comm_group) Characters_DirProds_iso:
  assumes "DirProds Gs I ≅ G" "group (DirProds Gs I)" "finite I"
  shows "DirProds (Characters ∘ Gs) I ≅ Characters G"
```

Using this lemma, the isomorphism of a finite cyclic group to its character group, and the *fundamental theorem of finitely generated abelian groups*, it is possible to derive the following for a finite abelian group G:

$$G \cong \mathbb{Z}_{d_1} \times \ldots \times \mathbb{Z}_{d_n} \cong \widehat{\mathbb{Z}_{d_1}} \times \ldots \times \widehat{\mathbb{Z}_{d_n}} \cong \widehat{G}$$

In Isabelle, this is captured in the following lemma:

```
lemma (in finite_comm_group) Characters_iso:
  shows "G ≅ Characters G"
```

It is worth noting that, in contrast to the natural isomorphism between a group and its double dual[2], the isomorphism to its dual is not that obvious. Even proving the existence of such an isomorphism required the use of the *fundamental theorem of finitely generated abelian groups*.

5.3.2 Several properties of characters

This subsection will cover – not in great detail – the efforts made to get rid of the theory `Adjoin_Groups`, as it appears rather ad-hoc and just not very elegant. Several properties of characters and character groups will be reproved using more standard group-theoretic arguments.

The main difficulty in this part was to reprove that the number of ways in which a character on a subgroup H can be extended to a character on the whole group G, is $\frac{|G|}{|H|}$. This expression has obvious similarity with Lagrange's (group-theoretic) theorem and this was the starting point for the proof. The following definition plays a central role:

[2]For $x \in G$, define $\hat{x} : \widehat{G} \to \widehat{\widehat{G}}$, $\hat{x}(\chi) := \chi(x)$. Then $f : G \to \widehat{\widehat{G}}$, $f(x) = \hat{x}$ is the natural isomorphism from G to its double dual.

```
definition restrict_char::"'a set ⇒ ('a ⇒ complex) ⇒ ('a ⇒ complex) " where
"restrict_char H χ = (λe. if e∈H then χ e else 0)"
```

`restrict_char H χ` restricts a given character χ to the specified subgroup H. When applied to characters of the whole group G, this operation actually is a group homomorphism from $\widehat{G}$ to $\widehat{H}$. Its image is all of $\widehat{H}$ and its kernel is the set of characters that project all elements of H onto 1:

```
lemma (in finite_comm_group) restrict_char_hom:
  assumes "subgroup H G"
  shows "group_hom (Characters G) (Characters (G⦇carrier := H⦈)) (restrict_char H)"

lemma (in finite_comm_group) restrict_char_kernel:
  assumes "subgroup H G"
  shows "kernel (Characters G) (Characters (G⦇carrier := H⦈)) (restrict_char H)
          = {χ∈characters G. ∀x∈H. χ x = 1}"

lemma (in finite_comm_group) restrict_char_image:
  assumes "subgroup H G"
  shows "restrict_char H ` (carrier (Characters G))
          = carrier (Characters (G⦇carrier := H⦈))"
```

As every character on H is the image of some character on G under restriction, it can be extended to a character on G. With the following lemma, it is further possible to reduce the question of the number of extensions of any character on H to the number of extensions of the principal character on H:

```
lemma (in finite_comm_group) character_restrict_card:
  assumes "subgroup H G" "character G a" "character G b"
  shows "card {χ'∈characters G. ∀x∈H. χ' x = a x}
        = card {χ'∈characters G. ∀x∈H. χ' x = b x}"
```

In order to obtain the number of extensions of the principal character on H to G, the following observation is crucial: the kernel of the `restrict_char` homomorphism from $\widehat{G}$ to $\widehat{H}$ (all characters that project all elements of H onto 1) is isomorphic to the character group of the quotient group $\widehat{G/H}$: By identifying an element x of G with its coset xH, every character on G/H (acting on cosets) is turned into a character on G that maps all elements of H onto 1. This mapping between $\widehat{G/H}$ and this restricted set of characters on G is an isomorphism:

```
lemma (in finite_comm_group) iso_Characters_FactGroup:
  assumes H: "subgroup H G"
  shows "(λχ x. if x ∈ carrier G then χ (H #> x) else 0) ∈
          iso (Characters (G Mod H))
            ((Characters G)⦇carrier := {χ∈characters G. ∀x∈H. χ x = 1}⦈)"
```

```
lemma (in finite_comm_group) is_iso_Characters_FactGroup:
  assumes H: "subgroup H G"
  shows "Characters (G Mod H)
    ≅ (Characters G)⦇carrier := {χ∈characters G. ∀x∈H. χ x = 1}⦈"
```

Putting all of this together, one obtains for a character χ on H[3]:

$$|\{\chi' \in \widehat{G}.\ \forall x \in H.\ \chi'(x) = \chi(x)\}| = |\{\chi' \in \widehat{G}.\ \forall x \in H.\ \chi'(x) = 1\}| = |\widehat{G/H}| = |G/H|$$

Together with Lagrange's theorem, it is then evident that:

```
theorem (in finite_comm_group) card_character_extensions:
  assumes "subgroup H G" "character (G⦇carrier := H⦈) χ"
  shows "card {χ'∈characters G. ∀x∈H. χ' x = χ x} * card H = order G"
```

Having reproved this, it follows immediately that one can extend a character χ on H to a character on G (by choosing one of the $\frac{|G|}{|H|}$ characters on G with the same values on H as χ)[4]:

```
corollary (in finite_comm_group) character_extension_exists:
  assumes "subgroup H G" "character (G⦇carrier := H⦈) χ"
  obtains χ' where "character G χ'" and "⋀x. x ∈ H ⟹ χ' x = χ x"
```

And lastly, it was possible to show that for every element $x \in G$, $x \neq \mathbf{1}$ there exists a character on G projecting this element on a $ord(x)$-th root of unity:

```
corollary (in finite_comm_group) character_with_value_exists:
  assumes "x ∈ carrier G" and "x ≠ 1" and "z ^ ord x = 1"
  obtains χ where "character G χ" and "χ x = z"
```

This can easily be seen when considering a character on the cyclic group generated by x together with the results of Listing 5.2 and then extending this character to the whole group G.

[3]One may come to the conclusion that the `restrict_char` homomorphism may not be needed at all as it does not show up in the chain of equations. However, this is not the case as its surjectivity is the key fact that allows the application of `character_restrict_card`. To see this, note that the preconditions of this lemma mention characters on the whole group G rather than on just H. And such a character on G can be obtained from the pre-image of `restrict_char`.

[4]It is also possible to derive this directly from the fact that `restrict_char` is a surjective homomorphism from G to H.

6 Conclusion

In this thesis, I successfully formalised the *fundamental theorem of finitely generated abelian groups* and subsequently used it to prove and reprove some of the results on *character groups*, producing around 5000 lines of formalisation code.

6.1 Lessons learned

Although I think of this thesis as a success, in retrospect, there is a lot of room for improvement: some of my early proofs are very verbose and could surely be rewritten in a more elegant and shorter way – also the proof of the *invariant factor decomposition* by using lists to express relations instead of sets.

Another important – and to some, rather obvious – lesson is to double check newly introduced definitions for correctness before starting to formalise facts about them. I started this formalisation work out with an incorrect generalisation of the binary *internal direct product*, which cost me some time.

Finally, when working in a proof assistant and strictly formalising each step, one might get caught up in this lengthy process and only think about the next small formalisation step. However, in order to work efficiently, it is crucial not to lose sight of the bigger context.

6.2 Future Work

As shown in *chapter 5*, the formalisation of the *fundamental theorem of finitely generated abelian groups* opens up new areas for formalisation – in group theory and other fields, such as analytic number theory. However, as already mentioned at the end of *subsection 4.3.1 Invariant factor decomposition*, in this work I was not able to prove the uniqueness of the *invariant factor decomposition* (and as a consequence also not of the *primary decomposition*). Proving this would finalise the formalisation work on the *fundamental theorem of finitely generated abelian groups.* But since all sources I found about the proof of the uniqueness of these decompositions make use of *torsion subgroups* and the *invariant basis number* property for $\mathbb{Z}$-modules, I suspect that one would have to formalise some theory on these subjects first. For now, it remains a task open for future work.

Another point is that, in the course of formalising the *fundamental theorem of finitely generated abelian groups* and the results on *character groups*, I also formalised a lot of lemmas, some of which – in my opinion – could make their way into the standard *HOL-Algebra* *library*. In such a case, another look at the lemmas – in order to meet standards for a library – might reveal more elegant proofs. Also, some documentation will have to be added.

And lastly, a more global view also reveals that there is still a lot to do in order to formalise entire group theory in Isabelle, let alone all known mathematics – a truly demanding but in parts necessary task in order to allow for new mathematical developments actually taking place in a theorem prover like Isabelle.

Bibliography

[1] J. Otal. “The Classification of the Finite Simple Groups: An Overview”. In: 2004.

[2] K. Gödel. “Über Formal Unentscheidbare Sätze der Principia Mathematica Und Verwandter Systeme I”. In: *Monatshefte für Mathematik* 38.1 (1931), pp. 173–198.

[3] M. Wenzel. *The Isabelle/Isar Reference Manual.* `https://isabelle.in.tum.de/doc/isar-ref.pdf`, Also distributed with Isabelle. Feb. 2021.

[4] `https://isabelle.in.tum.de`, Accessed May 10, 2021.

[5] M. Wenzel. “Isar — A Generic Interpretative Approach to Readable Formal Proof Documents”. In: *Theorem Proving in Higher Order Logics.* Ed. by Y. Bertot, G. Dowek, L. Théry, A. Hirschowitz, and C. Paulin. Berlin, Heidelberg: Springer Berlin Heidelberg, 1999, pp. 167–183. ISBN: 978-3-540-48256-7.

[6] J. Blanchette, M. Desharnais, and L. C. Paulson. *Hammering Away, A User’s Guide to Sledgehammer for Isabelle/HOL.* Distributed with Isabelle. 2021.

[7] `https://www.isa-afp.org/`, Accessed May 10, 2021.

[8] `https://www.isa-afp.org/statistics.html`, Accessed May 4, 2021.

[9] H. Kobayashi, L. Chen, and H. Murao. “Groups, Rings and Modules”. In: *Archive of Formal Proofs* (May 2004). `https://isa-afp.org/entries/Group-Ring-Module.html`, Formal proof development. ISSN: 2150-914x.

[10] F. Kammüller, M. Wenzel, and L. Paulson. “Locales A Sectioning Concept for Isabelle”. In: Sept. 1999. ISBN: 978-3-540-66463-5. DOI: `10.1007/3-540-48256-3_11`.

[11] C. Ballarin. *Tutorial to Locales and Locale Interpretation.* Published in L. Lambán, A. Romero, J. Rubio, editors, Contribuciones Científicas en honor de Mirian Andrés. Servicio de Publicaciones de la Universidad de La Rioja, Logroño, Spain, 2010. Also distributed with Isabelle. 2010.

[12] C. Ballarin, F. Kammüller, and L. Paulson. “The Isabelle/HOL Algebra Library”. In: (Feb. 2021).

[13] G. Gonthier, A. Asperti, J. Avigad, Y. Bertot, C. Cohen, F. Garillot, S. Le Roux, A. Mahboubi, R. O’Connor, S. O. Biha, I. Pasca, L. Rideau, A. Solovyev, E. Tassi, and L. Théry. “A Machine-Checked Proof of the Odd Order Theorem”. In: *Proceedings of the 4th International Conference on Interactive Theorem Proving.* ITP’13. Rennes, France: Springer-Verlag, 2013, pp. 163–179. ISBN: 9783642396335. DOI: `10.1007/978-3-642-39634-2_14`. URL: `https://doi.org/10.1007/978-3-642-39634-2_14`.

[14] J. Sylvestre. “Representations of Finite Groups”. In: *Archive of Formal Proofs* (Aug. 2015). `https://isa-afp.org/entries/Rep_Fin_Groups.html`, Formal proof development. ISSN: 2150-914x.

[15] M. Eberl. “Dirichlet L-Functions and Dirichlet’s Theorem”. In: *Archive of Formal Proofs* (Dec. 2017). `https://isa-afp.org/entries/Dirichlet_L.html`, Formal proof development. ISSN: 2150-914x.

[16] The mathlib Community. “The lean mathematical library”. In: *Proceedings of the 9th ACM SIGPLAN International Conference on Certified Programs and Proofs* (Jan. 2020). DOI: `10.1145/3372885.3373824`. URL: `http://dx.doi.org/10.1145/3372885.3373824`.

[17] A. Mahboubi and E. Tassi. *Mathematical Components*. Zenodo, Jan. 2021. DOI: `10.5281/zenodo.4457887`. URL: `https://doi.org/10.5281/zenodo.4457887`.

[18] `https://fm.mizar.org/`, Accessed May 12, 2021.

[19] `http://mmlquery.mizar.org/`, Accessed May 12, 2021.

[20] H. Okazaki, H. Yamazaki, and Y. Shidama. “Isomorphisms of Direct Products of Finite Commutative Groups”. Version 8.1.01 5.9.1172. In: *Formalized Mathematics* 21.1 (2013), pp. 65–74. ISSN: 1426-2630. DOI: `10.2478/forma-2013-0007`.

[21] C. Karpfinger and K. Meyberg. *Algebra*. Jan. 2017. ISBN: 978-3-662-54721-2. DOI: `10.1007/978-3-662-54722-9`.

[22] C. F. Gauß. *Disquisitiones arithmeticae*. Leipzig, 1801.

[23] G. Kemper. *Lecture notes of Algebra*. `https://www.groups.ma.tum.de/fileadmin/w00ccg/algebra/people/kemper/lectureNotes/Algebra.pdf`. Apr. 2020.

[24] E. S. Jan-Hendrik Evertse. *Lecture notes of Analytical number theory, Chapter 4*. `http://www.math.leidenuniv.nl/~evertse/ant16-4.pdf`. Sept. 2016.